VIEW와 MODEL의 목표

VIEW

VIEW 자신의 변화에 집중.
다른 VIEW나 MODEL에 직접 접근을 자제

MODEL

데이터의 getter, setter 역할로 끝
VIEW에 직접적인 접근을 자제

뭐가 view고 뭐가 model이냐?

1.view

역할이 다르면 각각 view로 분리.
지금은 이렇게 두 개로 나누기.

- blog post list를 표현하는 view
- 찜한 목록을 보여주는 view를 나누기.

2. model(store)

해당 view에서만 사용하는 data가 아니라면 별도 model(store)공간에 데이터를 보관.

VIEW와 MODEL을 조화롭게 이어주고 처리해주는 장치가 필요.

CONTROLLER

VIEW나 MODEL의 변화에따라 해야할 일을 등록

DISPATCHER

변화가 발생시 미리 가지고 있는 정보를 토대로, 필요할 일을 실행시킴.

> 일종의 Observer pattern을 사용해보자.

예제.

view에 어떠한 이벤트가 발생했을 때 그것을 알림.

뭐 이런식?

```
dispatcher.emit({
        'type' : "afterMove",
        'data' : [order]
});
```

그렇게 처리되기 위해서는 dispatcher에 콜백함수를 미리 등록

```
"afterMove" : function(order) {
        this.model.changeData(order);
}
```

dispatcher는 'afterMove'를 보관했다가, emit 실행되면 해당 함수를 실행.

즉,

dispatcher에 어떤 뷰에 어떤 일이 생기면 어떻게 처리하라고 등록.
(register)
어떤일이 실제로 발생하면 등록된 함수중 일치하는 녀석을 실행.(emit)

model의 변경 이후에도 emit실행할 수 있음.

model역시 필요하면 자신의 일 이후에 어떤 view의 변화가 필요하는 경우가 생길 수 있음.
그런경우 emit을 통해 자신의 변경 사실을 알림.

```
changeCurrentNews : function(order) {
        this.changeData = order;
        dispatcher.emit({
                "type" : "changeCurrentPanel",
                "data" :  [order]
    });
},
```

물론, 앞선 페이지의 changeCurrentPanel type도 미리 dispatcher에 등록돼 있어야 함.

아래는 dispatcher에 이미 등록된 콜백함수.

```
"changeCurrentPanel" : function (nextOrder) {
        this.listView.setHighLightTitle(nextOrder);
}.bind(this)
```

Dispatcher **구현은 어떻게?**

- 필요한 콜백을 등록해서 보관할 수 있어야하고 (register)
- 요청이 오면 등록된 콜백함수를 실행(emit)

super simple Dispatcher

```
class Dispatcher {
  register(fnlist) {
    this.fnlist = fnlist;
  }
  emit(o) {
    this.fnlist[o.type].apply(null, o.data);
  }
}
```

Example

간단한 예제로 dispatcher기반 개발 다시 이해.

http://jsbin.com/pocutapaxi/1/edit?html,js,output

Controller ?

dispatcher의 register는
view와 model, model과 view, view와 view의 관계를 어떤 type이라는 행위(action)로 표현한 셈.

이 부분을 controller라고 지칭.

```
dispatcher.register({
    "initView" : function(result) {
      this.model.saveAllNewsList(result);
    }.bind(this),

    "afterMove" : function(order) {
      this.model.changeData(order);
    },
    "changeCurrentPanel" : function (nextOrder) {
      this.listView.setHighLightTitle(nextOrder);
    }.bind(this)
    ....
```

project **코드 변경하기**.

위 개념을 적용해보기! 찜하기 부분에 적용하자.

- dispatcher를 만들고,
- 하나의 클래스에서 찜리스트를 별도 store클래스로 분리하고,
- view도 분리하고 (별도 클래스나 함수로)
- controller 도 별도 클래스로 분리해서 만들고.